DURÉE: 1h30, UNE FEUILLE A5 RECTO-VERSO MANUSCRITE AUTORISÉE

REMARQUES:
- Les exercices sont indépendants et peuvent être réalisés dans l'ordre voulu.
- Dans l'implémentation d'une méthode, on pourra utiliser n'importe quelle autre méthode définie auparavant même si celle-ci n'a pas été implémentée.
- Dans toutes les implémentations que vous écrivez, pensez à respecter le guide de syntaxe pour la programmation (règles de nommage, présentation des blocs, *etc.*).

EXERCICE 1: ALGORITHMIE (6 POINTS)

1. Écrire une fonction `combienDeMots( chaine )` qui compte le nombre de mots dans une chaîne de caractères.
   - On supposera que les mots sont séparés par des espaces.
   - La chaîne pourra contenir les caractères de ponctuation suivant : `,.!?;:'"-` et on s'assurera donc qu'un mot contient toujours au moins une lettre.
     Par exemple, la phrase `Einstein disait : " Deux choses sont infinies : l'Univers et la bêtise humaine. "` contient 11 mots.
   - On rappelle qu'il existe une fonction `isalpha` qui vérifie si tous les caractères d'une chaine sont des lettres.

Solution: (1,5 points) : les solutions des questions sont "optimisées". On acceptera évidemment des solutions en plusieurs lignes.
0,5 point pour la suppression de la ponctuation dans la phrase, 1 point pour le comptage

```
def combienDeMots( phrase ) :
  mots = phrase.split( )
  contientLettres = [ sum( 1 for l in m if l.isalpha( ) ) for m in mots ]   #
     nombre de lettres (ponctuation exclue) par element du tableau
  nbMots = sum( 1 for c in contientLettres if c > 0 ) # si c > 0, il y a au
     moins une lettre dans l'element du tableau => c'est un mot
  return nbMots
```

2. Écrire une fonction `isStrOk( phrase )` qui renvoie un booléen indiquant si une chaîne de caractères ne contient que des lettres ou des caractères de ponctuation. La phrase ne devra pas être modifiée.

Solution: (1,5 points) : 0,5 point si la chaîne n'est pas modifiée, 1 point pour le test

```
def isStrOk( phrase ) :
  phrase2 = phrase           # on copie la phrase pour eviter les modifications
  for c in " ,.!?-;:'\"" :  # on supprime les caracteres de ponctuation
    phrase2 = phrase2.replace( c, "" )
  return phrase2.isalpha( ) # on verifie que la chaine restante ne contient que
      des lettres
```

3. Écrire une fonction `str2Tab( phrase )` qui renvoie un tableau contenant chaque lettre d'une chaîne de caractères. Les lettres devront toutes être en minuscule et les caractères de ponctuation ne devront pas apparaître dans le tableau. La phrase ne devra pas être modifiée.

Solution: (1 point)

```
def str2Tab( phrase ) :
  tab = [ c for c in phrase2.lower( ) if c.isalpha( ) ] # on convertit la
     chaine en tableau en ne conservant que les lettres
  """ OU """
  phrase2 = phrase           # on copie la phrase pour eviter les modifications
  for c in " ,.!?-;:'\"" :  # on supprime les caracteres de ponctuation
    phrase2 = phrase2.replace( c, "" )
  tab = [ c for c in phrase2.lower( ) ] # on convertit la chaine en tableau
  return tab
```

4. Écrire une fonction `isAnagramme( phrase1, phrase2 )` qui renvoie un booléen indiquant si les 2 chaînes de caractères passées en entrée sont des anagrammes.
   Un anagramme est un mot ou une phrase constitué des mêmes lettres qu'un autre mot ou phrase. Par exemple *Carpe Diem* et *Ça déprime* ou *Tom Marvolo Riddle* et *I am Lord Voldemort* sont des anagrammes. Si l'une des phrases n'est pas uniquement constituée de lettres ou de caractères de ponctuation, on renverra faux. On supposera aussi qu'aucun caractère accentué n'est présent.
   On rappelle qu'il existe en Python une fonction `sorted(tab)` qui renvoie une version triée d'un tableau `tab` (sans le modifier).

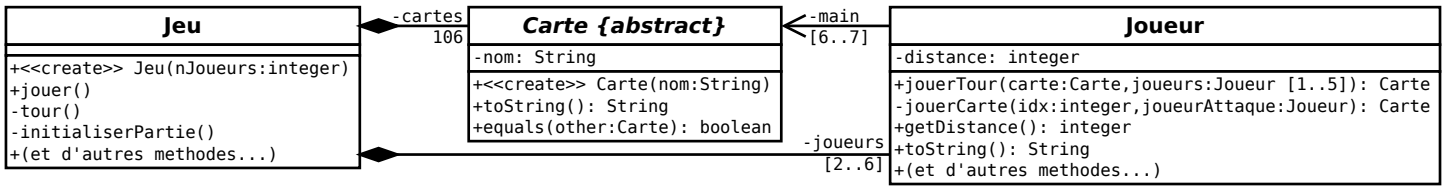
Solution: (1 point)
```
def isAnagramme( phrase1, phrase2 ) :
  if not( isStrOk( phrase1 ) and isStrOk( phrase2 ) ) :   # On verifie si les
     chaines sont correctes
    return False
  # On convertit les chaines en tableau et on trie les caracteres
  tab1, tab2 = sorted( str2Tab( phrase1 ) ), sorted( str2Tab( phrase2 ) )
  return tab1 == tab2   # On compare les 2 tableaux
```

5. Écrire un programme principal qui :
   — déclare 2 chaînes de caractères "Le boson scalaire de Higgs" et "L'horloge des anges ici-bas",
   — compte et affiche le nombre de mots de la deuxième chaîne de caractères,
   — vérifie et affiche (de manière claire) si les 2 chaînes de caractères sont des anagrammes

Solution: (1 point)
```
str1, str2 = "Le boson \"scalaire\" de Higgs", "L'horloge des anges ici-bas"
print( combienDeMots( str2 ) )
res = isAnagramme( str1, str2 )
print( "{} {} un anagramme de {}".format( str1, "est" if res else "n'est pas",
    str2 ) )
```

EXERCICE 2: "1000 BORNES™" (6 POINTS)

Le diagramme UML ci-dessous présente une modélisation partielle d'un jeu de "1000 Bornes™" .



Dans la classe `Jeu`, la méthode `jouer` distribue les Cartes aux Joueurs (créés au moment de la construction du `Jeu`) puis les fait jouer à tour de rôle jusqu'à ce qu'il y ait un vainqueur. La méthode `tour` permet d'effectuer un tour de jeu (*i.e.* de faire jouer une fois chaque joueur).

1. Pourquoi la classe `Carte` est-elle abstraite alors qu'aucune de ses méthodes ne l'est ?

Solution: (1 point) : Classe abstraite car elle sera spécialisée (Distance, Attaque, Défense) et on jouera avec ses héritiers. Il ne faut donc pas pouvoir la construire ⇒ classe abstraite

2. Quelle est la relation entre la classe `Jeu` et la classe `Carte` ? Justifier ce choix.

Solution: (1 point) : C'est une composition puisque le Jeu possède (sans partage) la Carte et que la Carte est détruite en même temps que le Jeu. Une Carte est une partie du Jeu.

3. Pourquoi la relation entre `Joueur` et `Carte` n'est-elle pas la même que celle entre `Jeu` et `Carte` ?

Solution: (1 point) : Un Joueur ne fait qu'utiliser des Carte le temps d'une partie. Elle ne lui appartiennent pas, ne sont pas créées ou détruites avec le Joueur, *etc.*⇒ il s'agit d'une association

4. Pourquoi la méthode `jouer` de `Jeu` est-elle publique alors que la méthode `tour` est privée ?

Solution: (1 point) :
Il faut pouvoir lancer la partie depuis l'extérieur de la classe ⇒ `jouer` doit être publique.
En revanche, il ne faut pas pouvoir lancer de tour de jeu depuis l'extérieur (c'est la méthode `jouer` qui s'en charge) sinon cela permettrait au 1er joueur de tricher (en lançant la méthode tour alors que ce n'est pas à lui pour gagner plus vite) ⇒ méthode privée

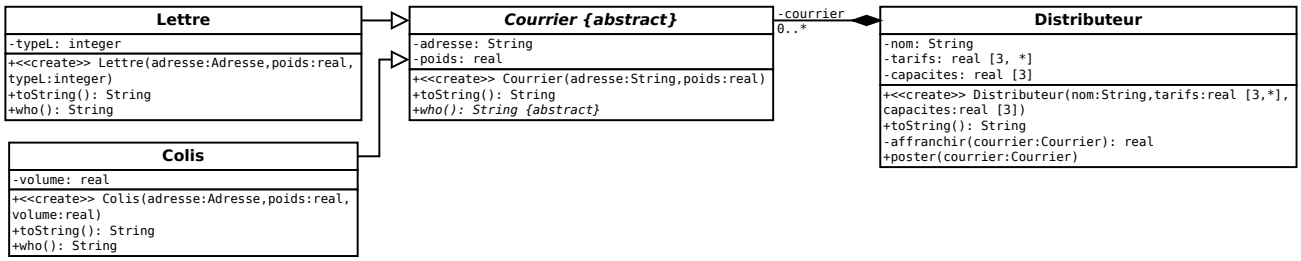
5. Donner l'implémentation de la méthode `+jouer( )`. Cette méthode peut être implémentée en appelant uniquement les méthodes présentes dans le diagramme UML. La méthode affichera l'état des`Joueurs` au début de chaque tour et affichera le vainqueur à la fin de la partie.

Solution: (2 points) : 1 point pour l'initialisation et la fin de partie, 1 point pour la gestion des tours de jeu

```
def jouer( self ) :
  # Debut de partie
  self.__initialiserPartie( )    # On initialise la partie
  bestJ = 0                      # Plus grande distance parcourue
  # Partie
  while bestJ < 1000 :
    # Etat des joueurs
    print( "-------------" )
    print( "\n".join( str(j) for j in sorted( self.joueurs, reverse=True ) ) )
    print( "-------------" )
    # On joue le tour
    self.__tour( )
    # On recupere la plus grande distance parcourue
    bestJ = max( [ j.getDistance( ) for j in self.joueurs ] )
  # Fin de partie
  allDist = [ j.getDistance( ) for j in self.joueurs ]  # Distances parcourues
  idxWinner = allDist.index( max( allDist ) )
  print( "-------------" )
  print( self.joueurs[idxWinner], "a gagne!" )
```

EXERCICE 3: PROGRAMMATION ORIENTÉE OBJET (10 POINTS)

Le diagramme UML ci-dessous modélise une application simplifiée d'envoi du courrier.



Un `Courrier` sera représenté par l'adresse du destinataire et son poids. La méthode `who( )` renverra le nom de la classe. L'attribut `typeL` de `Lettre` pourra être de type "normal" (0), "prioritaire" (1) ou avec accusé de réception AR (2). Pour un `Colis`, on conservera son volume en plus de son poids.
Un `Distributeur` sera modélisé par son nom (LaPoste, DHL, ...), les tarifs qu'il applique, les capacités avant envoi de sa flotte et les courriers qu'il doit distribuer. On considèrera que l'envoi d'un courrier passe par 2 étapes : il est d'abord posté (et payé) par le client puis lorsque l'une des capacités du distributeur est dépassée, l'ensemble du courrier reçu par celui-ci est envoyé. Les 3 lignes de l'attribut `tarifs` contiennent respectivement :

1. 2 cases : le prix au kilo pour une lettre ou un colis

2. les prix en fonction du type de la lettre (normal, prioritaire ou avec AR)
3. les prix au m^3 pour des colis de moins de $0.01m^3$, de moins de $0.25m^3$ ou de plus de $0.25m^3$.

L'attribut `capacites` est un tableau de 3 cases contenant le nombre (case 1), le poids (case 2) et le volume (case 3) maximaux de courriers. Dès que l'un de ces 3 critères n'est plus respecté, l'ensemble des courriers reçus est distribué.
Un exemple d'affichage (correspondant au programme principal de la question 3) est donné ci-dessous et on veillera à respecter ces consignes.

```
Lettre destine(e) a Chuck Norris, 5 place Jussieu, 75005 Paris // Poids: 0.17kg, Type: normal
Colis destine(e) a Darth Vador, 1ere a droite, Une galaxie fort fort lointaine // Poids: 7.55kg,
    Volume: 1.70m^3
Lettre destine(e) a Hairy Otter, Quai 9 3/4, 87160 Arnac-la-Poste // Poids: 0.30kg, Type: AR
Courrier poste. Prix: 122.63 euros
Courrier poste. Prix: 5.59 euros
Courrier poste. Prix: 0.34 euros
DHL: 2 lettres et 1 colis pour un poids de 8.01kg et un volume de 1.70m^3.
```

1. **Classes** `Courrier`, `Lettre` **et** `Colis` :
   (a) Implémenter la classe `Courrier` (import, attributs, méthodes).

Solution: (1,5 points) : 0.5 pour l'import, 0.5 pour les méthodes abstraites

```
from abc import ABC, abstractmethod
class Courrier (ABC) :
  def __init__( self, destinataire, poids ) :
    self.destinataire, self.poids = destinataire, poids
  def __str__( self ) :
    return "{} destine(e) a {} // Poids: {}kg".format( self.who( ), self.
        destinataire, self.poids )
  @abstractmethod
  def who( ) :  pass
```

   (b) Pourquoi la méthode `who` est-elle abstraite ? Quelle(s) conséquence(s), cela a t'il sur la classe `Courrier` ?

Solution: (1 point)
C'est une méthode dont la classe `Courrier` ne connaît pas le comportement (car elles seront implémentées dans les classes filles). Il faut donc les déclarer abstraites.
Comme la classe `Courrier` possède une méthode abstraite, la classe est abstraite et ne peut pas être instanciée.

   (c) Implémenter la classe `Lettre`.

Solution: (1 point) : 0.5 pour l'utilisation de `super`

```
class Lettre (Courrier) :
  def __init__( self, destinataire, poids, typeL ) :
    super( ).__init__( destinataire, poids )
    self.typeL = typeL
  def __str__( self ) :
    return "{}, Type: {}".format(super().__str__(), ["normal", "prioritaire"
        , "AR"][self.typeL])
  def who( self ):    return "Lettre"
```

2. **Classe** `Distributeur` :
   (a) Implémenter la déclaration de la classe `Distributeur` ainsi que le constructeur et la méthode `toString`. On affichera le nom, le nombre de lettres, le nombre de colis, le poids total des courriers à envoyer ainsi que le volume des colis.

Solution: (1,5 points)

```
class Distributeur :
  def __init__( self, nom, tarifs, capacites ) :
    self.nom = nom
    self.tarifs, self.capacites = tarifs, capacites
    self.courrier = [ ]
  def __str__( self ) :
    nL, nC = 0, 0
    tP, tV = 0, 0
    for c in self.courrier :
      tP += c.poids
      if isinstance( c, Lettre ) :
        nL += 1
      elif isinstance( c, Colis ) :
        nC += 1
        tV += self.volume
    res = "{}: {} lettres et {} colis pour un poids de {}kg et un volume de
       {}m^3.".format( self.nom, nL, nC, tP, tV )
    return res
```

(b) Implémenter la fonction `-affranchir( courrier: Courrier ): real`, qui renvoie le prix pour envoyer le `courrier`. Le prix sera calculé de la manière suivante :
— pour une `Lettre` : (prix au kilo d'une lettre * poids de la lettre) + prix en fonction du type de la lettre
— pour un `Colis` : (prix au kilo d'un colis * poids du colis) + (prix au m^3 * volume du colis)

Solution: (1,5 points) : 1 point pour l'utilisation correcte d'isinstance

```
def __affranchir( self, courrier ) :
  if isinstance( courrier, Lettre ) :
    res = self.tarifs[0][0]*courrier.poids + self.tarifs[1][courrier.typeL]
  elif isinstance( courrier, Colis ) :
      res = self.tarifs[0][1] * courrier.poids
    if courrier.volume < 0.01 :
      res += self.tarifs[2][0] * courrier.volume
    elif courrier.volume < 0.25 :
      res += self.tarifs[2][1] * courrier.volume
    else:
      res += self.tarifs[2][2] * courrier.volume
  else :       # Securite: si courrier n'est ni une Lettre ni un Colis
    res = -1   # Ne pas penaliser si oubli
  return res
```

(c) Implémenter la fonction `+poster( courrier: Courrier )`, qui
— affiche un message indiquant que le `courrier` est reçu et le prix pour son envoi
— l'ajoute au courrier déjà reçu
— envoie l'ensemble du courrier si l'une des `capacites` est dépassée

Solution: (1,5 points) : 0.5 pour les étapes 1 et 2 ; 1 point pour l'envoi du courrier (calcul du poids et volume, RàZ du courrier)

```
def poster( self, courrier ) :
  prix = self.__affranchir( courrier )
  print( "Courrier poste. Prix: {} euros".format( prix ) )
  self.courrier.append( courrier )

  poidsTot = sum( c.poids for c in self.courrier )
  volTot = sum( c.volume for c in self.courrier if isinstance( c, Colis ) )
  if len( self.courrier ) > self.capacites[0] or \
    poidsTot > self.capacites[1] or volTot > self.capacites[2] :
    print( "Envoi du courrier!" )
    self.courrier = [ ]
```

3. **Programme principal** : écrire un programme principal qui génère des `Courriers` et les ajoute dans un tableau, affiche l'ensemble des `Courriers`, les poste auprès d'un `Distributeur` puis affiche le `Distributeur`.

Pour la génération des Courriers
— on effectuera un tirage aléatoire pour déterminer le type (`Lettre` ou `Colis`) de `Courrier`,
— une fonction `genererAdresse( ) : String` existe et permettra d'obtenir l'adresse du destinataire,
— on génèrera aléatoirement le poids (entre 0.1 et 0.5kg) et le type (normal, ...) d'une `Lettre`,
— on génèrera aléatoirement le poids (entre 0.5 et 10kg) et le volume (entre 0.005 et 2m^3) d'un `Colis`

Solution: (2 points) : 1 pour les import et la génération aléatoire de nombres, 0.5 point pour la génération des courriers, 0.5 pour le reste

```
from random import random, randint
c = [ ]
for _ in range( randint( 5, 10 ) ):
  dest = genererAdresse( )
  if randint( 0, 1 ) :
    poids, typeL = random( ) * 0.4 + 0.1, randint( 0, 2 )
    ci = Lettre( dest, poids, typeL )
  else :
    poids, volume = random( ) * 9.5 + 0.5, random(  ) * 1.995 + 0.005
    ci = Colis( dest, poids, volume )
  c.append( ci )
print( "- {}".format( "\n".join( str( ci ) for ci in c ) ) )

tarifs = [ [2, 5], [0, 2, 5], [10, 20, 50] ]    # Poids, Type de Lettre, Volume
    Colis
capacites = [10, 10, 2]
dhl = Distributeur( "DHL", tarifs, capacites )
for ci in c :
  dhl.poster( ci )
print( dhl )
```